Georgia Tech

DATA ANALYTICS USING DEEP LEARNING

GT 8803 // FALL 2019 // JOY ARULRAJ

LECTURE #03: LOSS FUNCTIONS & OPTIMIZATION

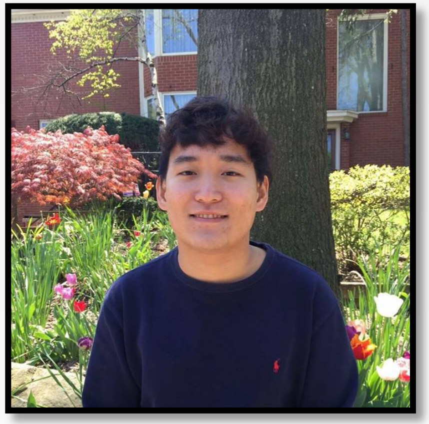
CREATING THE NEXT®

ADMINISTRIVIA

- Assignment 0
  - One-page overview of your background
  - Submit via Gradescope
  - Due on Wednesday

TEACHING ASSISTANTS

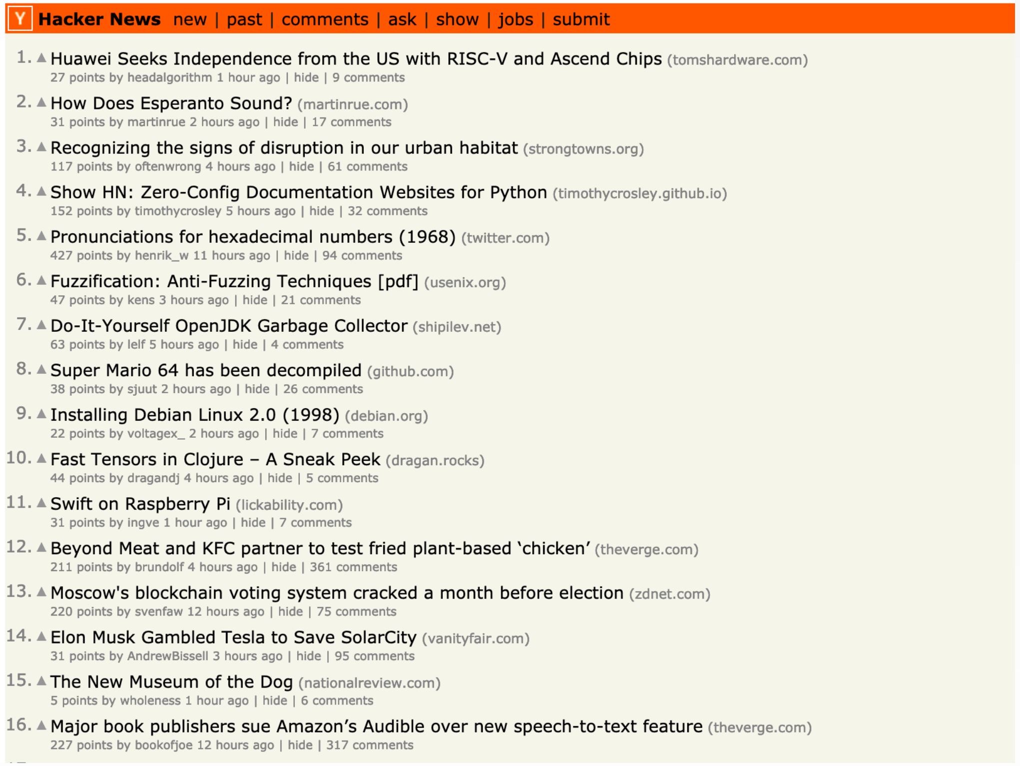
- TA #1: Jaeho Bang
  - Ph.D. student in CS
  - B.S. from Carnegie Mellon
  - Office hours: Tue 3:30 – 4:30 PM



- TA #2: Gaurav Tarlok Kakkar
  - M.S. student in CS
  - B.S. from IIT Kanpur
  - Office hours: Thu/Fri 3:00 – 4:00 PM

HACKER NEWS

- https://news.ycombinator.com/



Hacker News new | past | comments | ask | show | jobs | submit

1. ▲ Huawei Seeks Independence from the US with RISC-V and Ascend Chips (tomshardware.com)
   27 points by headalgorithm 1 hour ago | hide | 9 comments
2. ▲ How Does Esperanto Sound? (martinrue.com)
   31 points by martinrue 2 hours ago | hide | 17 comments
3. ▲ Recognizing the signs of disruption in our urban habitat (strongtowns.org)
   117 points by oftenwrong 4 hours ago | hide | 61 comments
4. ▲ Show HN: Zero-Config Documentation Websites for Python (timothycrosley.github.io)
   152 points by timothycrosley 5 hours ago | hide | 32 comments
5. ▲ Pronunciations for hexadecimal numbers (1968) (twitter.com)
   427 points by henrik_w 11 hours ago | hide | 94 comments
6. ▲ Fuzzification: Anti-Fuzzing Techniques [pdf] (usenix.org)
   47 points by kens 3 hours ago | hide | 21 comments
7. ▲ Do-It-Yourself OpenJDK Garbage Collector (shipilev.net)
   63 points by lelf 5 hours ago | hide | 4 comments
8. ▲ Super Mario 64 has been decompiled (github.com)
   38 points by sjuut 2 hours ago | hide | 26 comments
9. ▲ Installing Debian Linux 2.0 (1998) (debian.org)
   22 points by voltagex_ 2 hours ago | hide | 7 comments
10. ▲ Fast Tensors in Clojure – A Sneak Peek (dragan.rocks)
    44 points by dragandj 4 hours ago | hide | 5 comments
11. ▲ Swift on Raspberry Pi (lickability.com)
    31 points by ingve 1 hour ago | hide | 7 comments
12. ▲ Beyond Meat and KFC partner to test fried plant-based ‘chicken’ (theverge.com)
    211 points by brundolf 4 hours ago | hide | 361 comments
13. ▲ Moscow's blockchain voting system cracked a month before election (zdnet.com)
    220 points by svenfaw 12 hours ago | hide | 75 comments
14. ▲ Elon Musk Gambled Tesla to Save SolarCity (vanityfair.com)
    31 points by AndrewBissell 3 hours ago | hide | 95 comments
15. ▲ The New Museum of the Dog (nationalreview.com)
    5 points by wholeness 1 hour ago | hide | 6 comments
16. ▲ Major book publishers sue Amazon’s Audible over new speech-to-text feature (theverge.com)
    227 points by bookofjoe 12 hours ago | hide | 317 comments

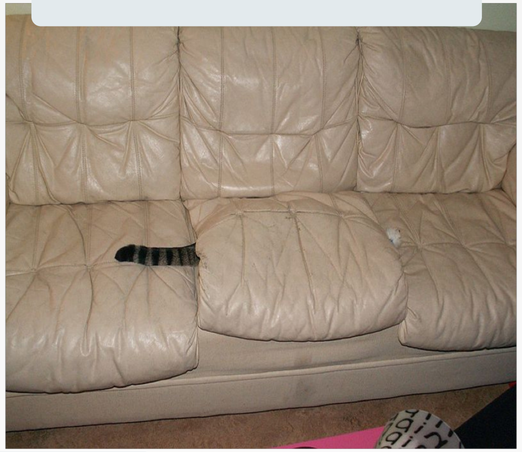
LAST CLASS: CHALLENGES OF RECOGNITION

VIEWPOINT

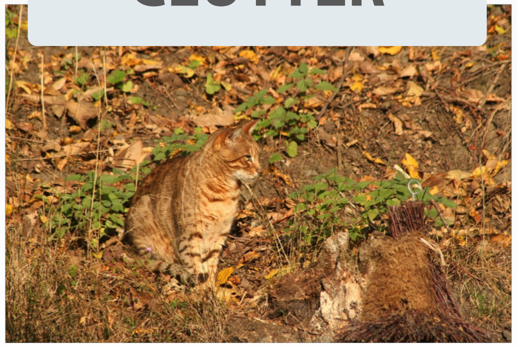
ILLUMINATION

DEFORMATION

OCCLUSION

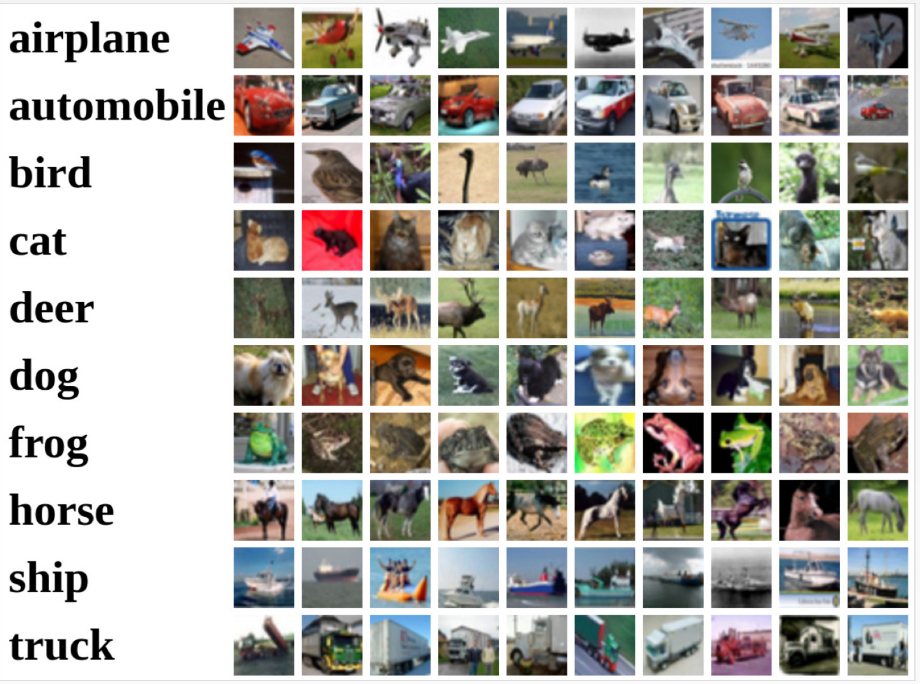


CLUTTER

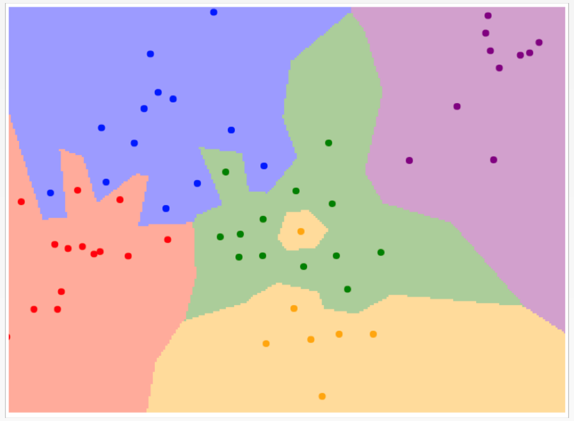


INTRACLASS VARIATION

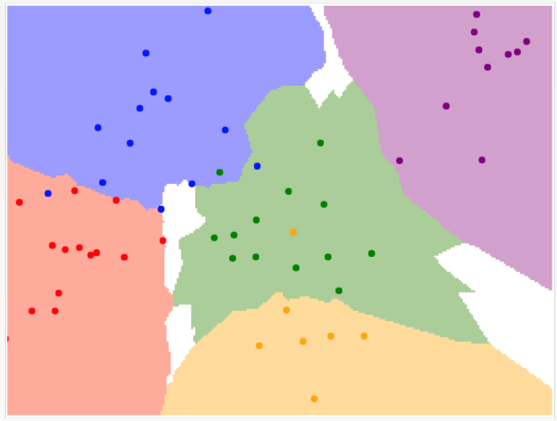
LAST CLASS: KNN, DATA-DRIVEN APPROACH

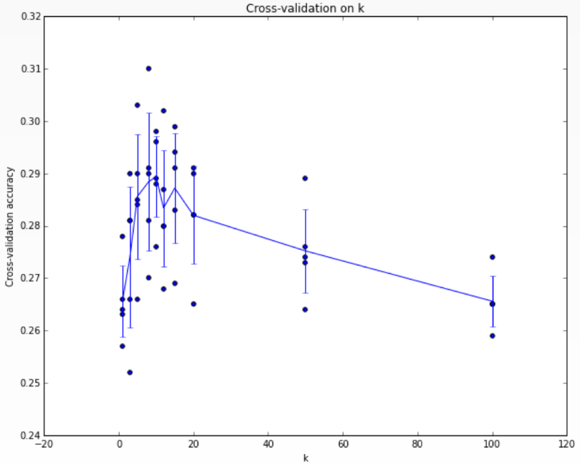


1-NN classifier

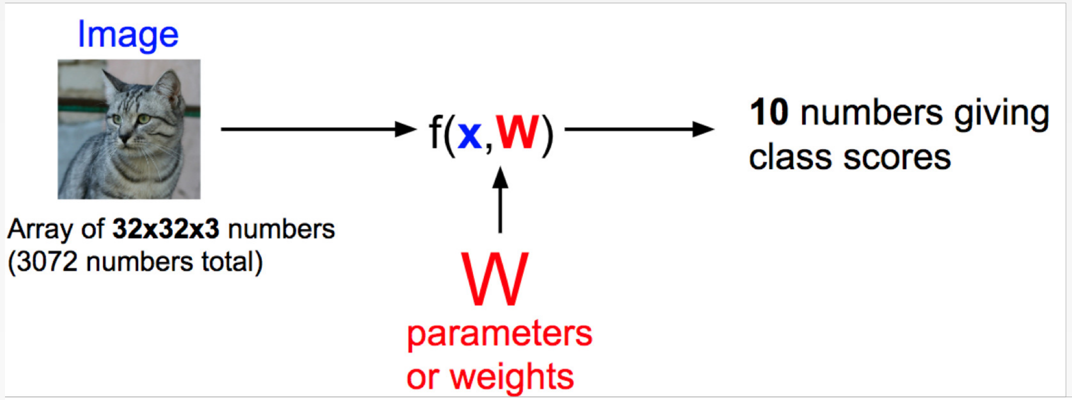


5-NN classifier





LAST CLASS: LINEAR CLASSIFIER



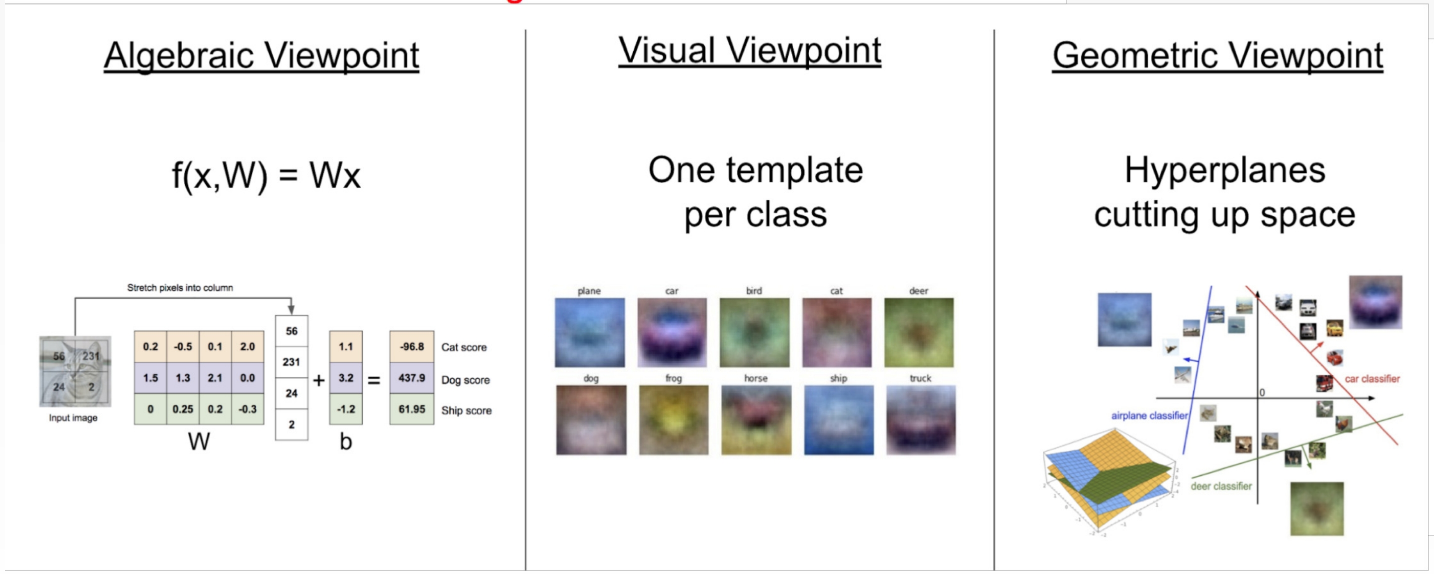
Image

Array of **32x32x3** numbers
(3072 numbers total)

10 numbers giving
class scores

W
parameters
or weights

$$f(x,W) = Wx + b$$



Algebraic Viewpoint

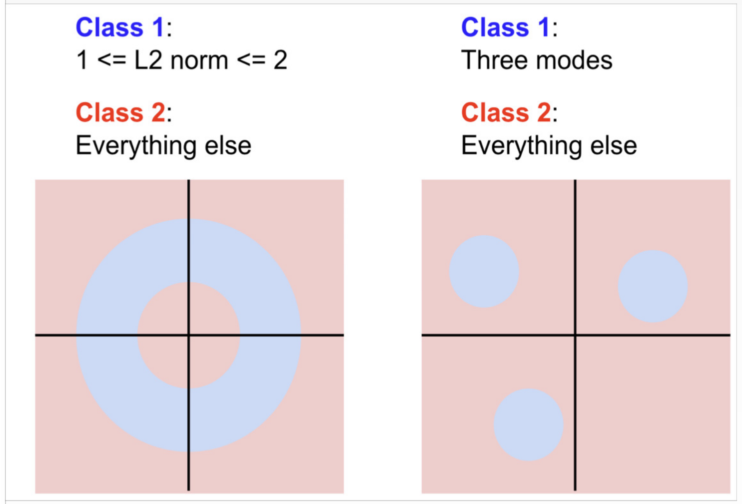
$f(x,W) = Wx$

Visual Viewpoint

One template
per class

Geometric Viewpoint

Hyperplanes
cutting up space



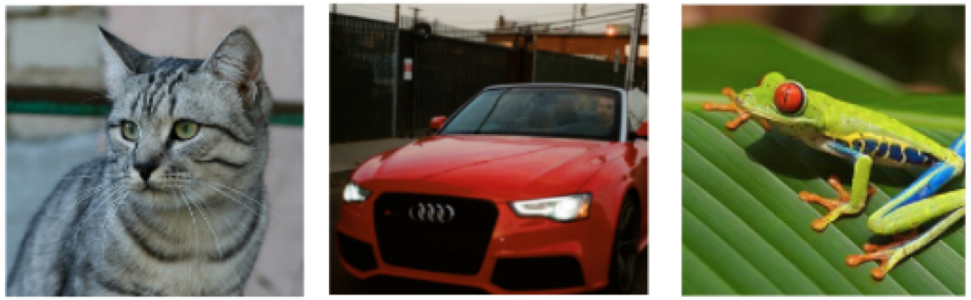
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else

Class 1:
Three modes

Class 2:
Everything else

LAST CLASS: LINEAR CLASSIFIER



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9** ✓	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14 ✗

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the **training data**.
2. Come up with a way of efficiently finding the **parameters** that **minimize** the loss function. **(optimization)**

TODAY'S AGENDA

- Loss functions
  - Multi-class SVM loss function
  - Regularization
  - Softmax loss function
- Optimization
  - Gradients
  - Stochastic Gradient Descent

LOSS FUNCTIONS

LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Loss Function:

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^N$$

Where x_i is image and
y_i is (integer) label

Loss over the dataset is a average of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

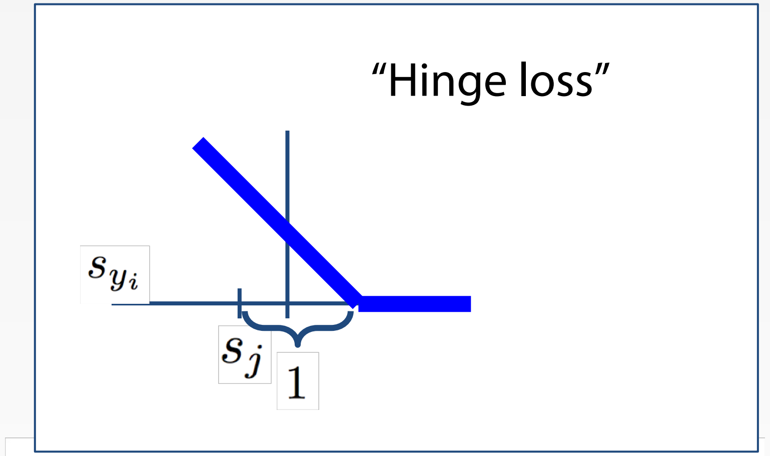
$$
\begin{aligned}
L_i &= \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases} \\
&= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)
\end{aligned}
$$

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass SVM loss:



$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x,W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

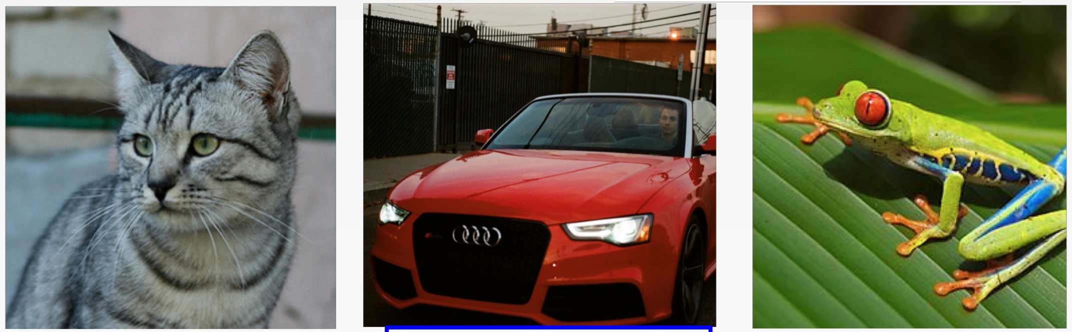
the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

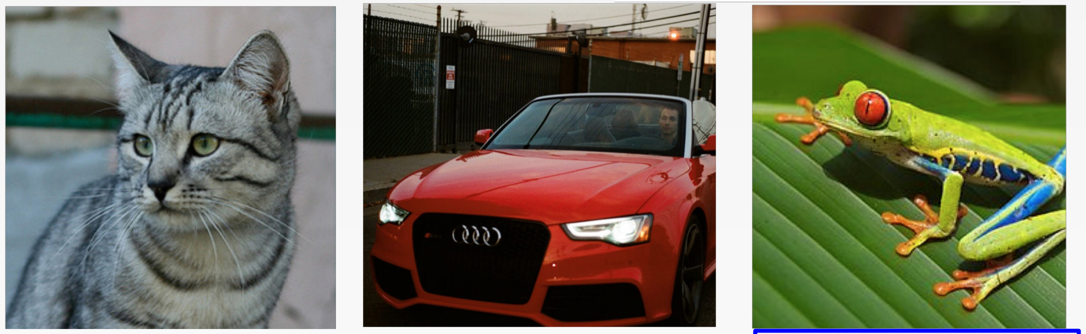
the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 6.3) + max(0, 6.6)
= 6.3 + 6.6
= 12.9

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Loss over full dataset is average:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

L = (2.9 + 0 + 12.9)/3 = **5.27**

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Q: What happens to loss if the scores for the car image change a bit?

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Q2: What is the min/max possible loss?

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x,W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Q3: At initialization W is small so all s ≈ 0.
What is the loss?

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Q4: What if the sum was over all classes? (including j = y_i)

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Q5: What if we used mean instead of sum?

MULTICLASS SVM LOSS FUNCTION

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x,W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

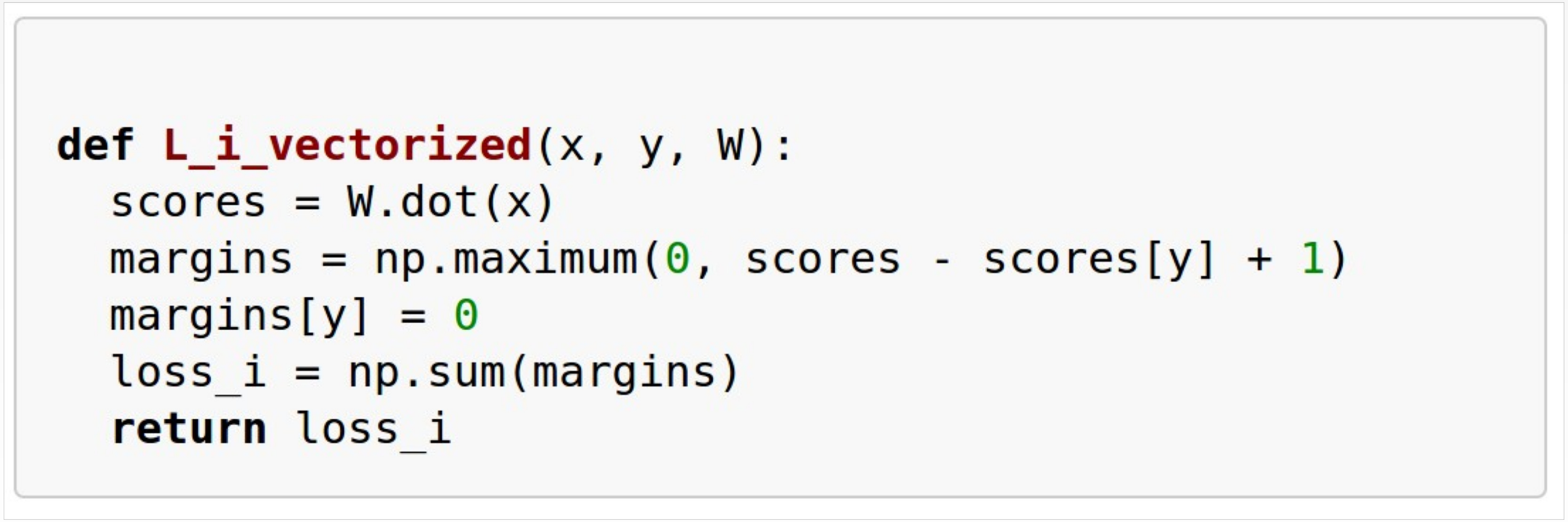
$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Q6: What if we used

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

MULTICLASS SVM LOSS FUNCTION

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



```
def L_i_vectorized(x, y, W):
  scores = W.dot(x)
  margins = np.maximum(0, scores - scores[y] + 1)
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i
```

MULTICLASS SVM LOSS FUNCTION

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

- Suppose that we found a W such that L = 0.
- Is this W unique?

MULTICLASS SVM LOSS FUNCTION

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

- Suppose that we found a W such that L = 0.
- Is this W unique?
- **No! 2W is also has L = 0!**

MULTICLASS SVM LOSS FUNCTION

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Before:

$= \max(0, 1.3 - 4.9 + 1)$
$\quad + \max(0, 2.0 - 4.9 + 1)$
$= \max(0, -2.6) + \max(0, -1.9)$
$= 0 + 0$
$= 0$

With W twice as large:

$= \max(0, 2.6 - 9.8 + 1)$
$\quad + \max(0, 4.0 - 9.8 + 1)$
$= \max(0, -6.2) + \max(0, -4.8)$
$= 0 + 0$
$= 0$

MULTICLASS SVM LOSS FUNCTION

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

- Suppose that we found a W such that L = 0.
- Is this W unique?
- **No! 2W is also has L = 0!**
- **How do we choose between W and 2W?**

REGULARIZATION

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}$$

Data loss: Model predictions should match training data

REGULARIZATION

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{\textbf{Data loss}: Model predictions should match training data}} + \underbrace{\lambda R(W)}_{\text{\textbf{Regularization}: Prevent the model from doing \textit{too} well on training data}}$$

REGULARIZATION

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)} + \underbrace{\lambda R(W)}$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

REGULARIZATION

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{} + \underbrace{\lambda R(W)}_{}$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

REGULARIZATION

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{} + \underbrace{\lambda R(W)}_{}$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

More complex:

Dropout

Batch normalization

Stochastic depth, etc.

REGULARIZATION

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{} + \underbrace{\lambda R(W)}_{}$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Why regularize?

- Express preferences over weights
- Make the model *simple* so it works on test data
- Improve optimization by adding curvature

REGULARIZATION: EXPRESSING PREFERENCES

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

REGULARIZATION: EXPRESSING PREFERENCES

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = \boxed{[0.25, 0.25, 0.25, 0.25]}$$

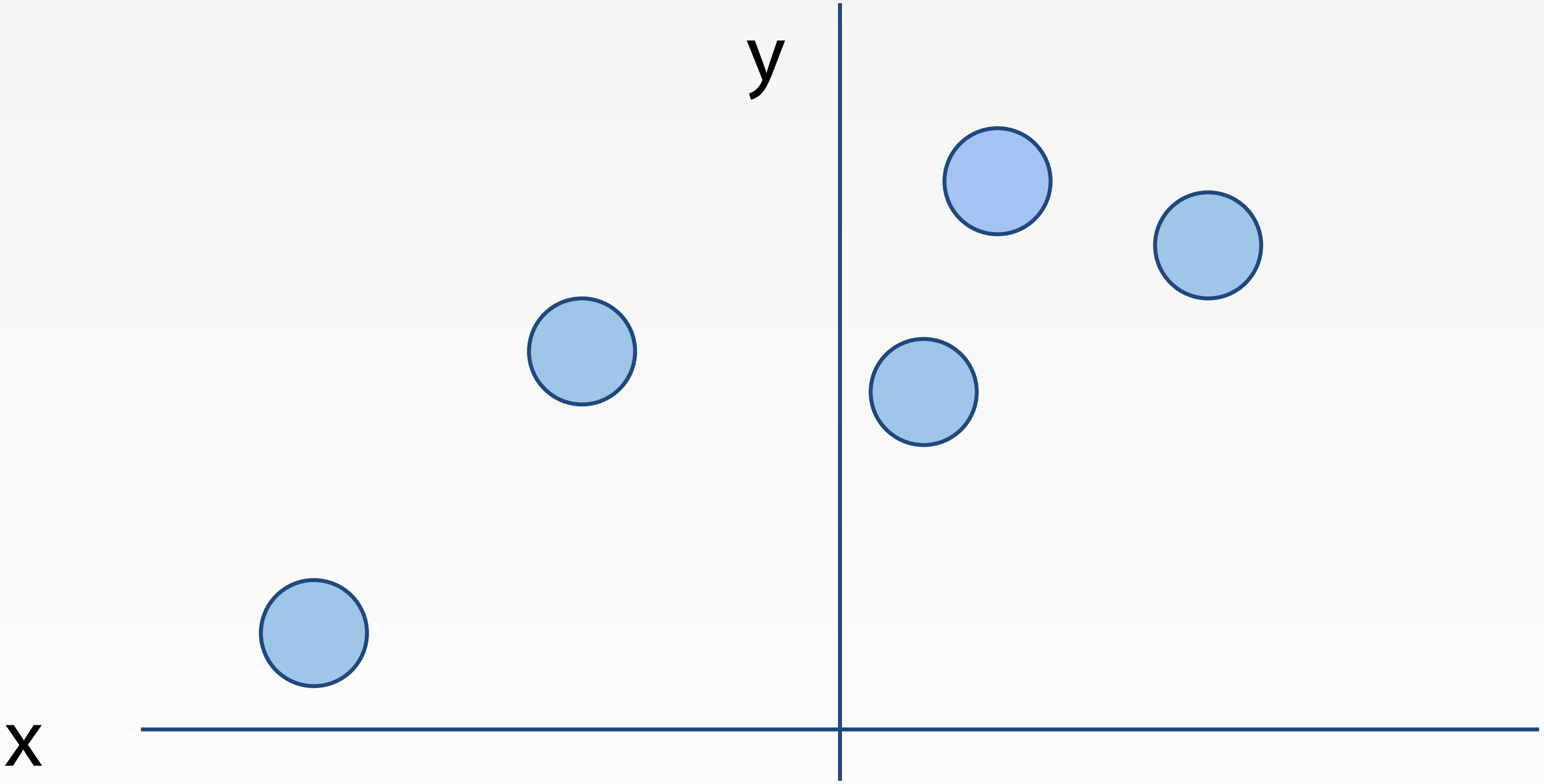
$$w_1^T x = w_2^T x = 1$$

L2 Regularization

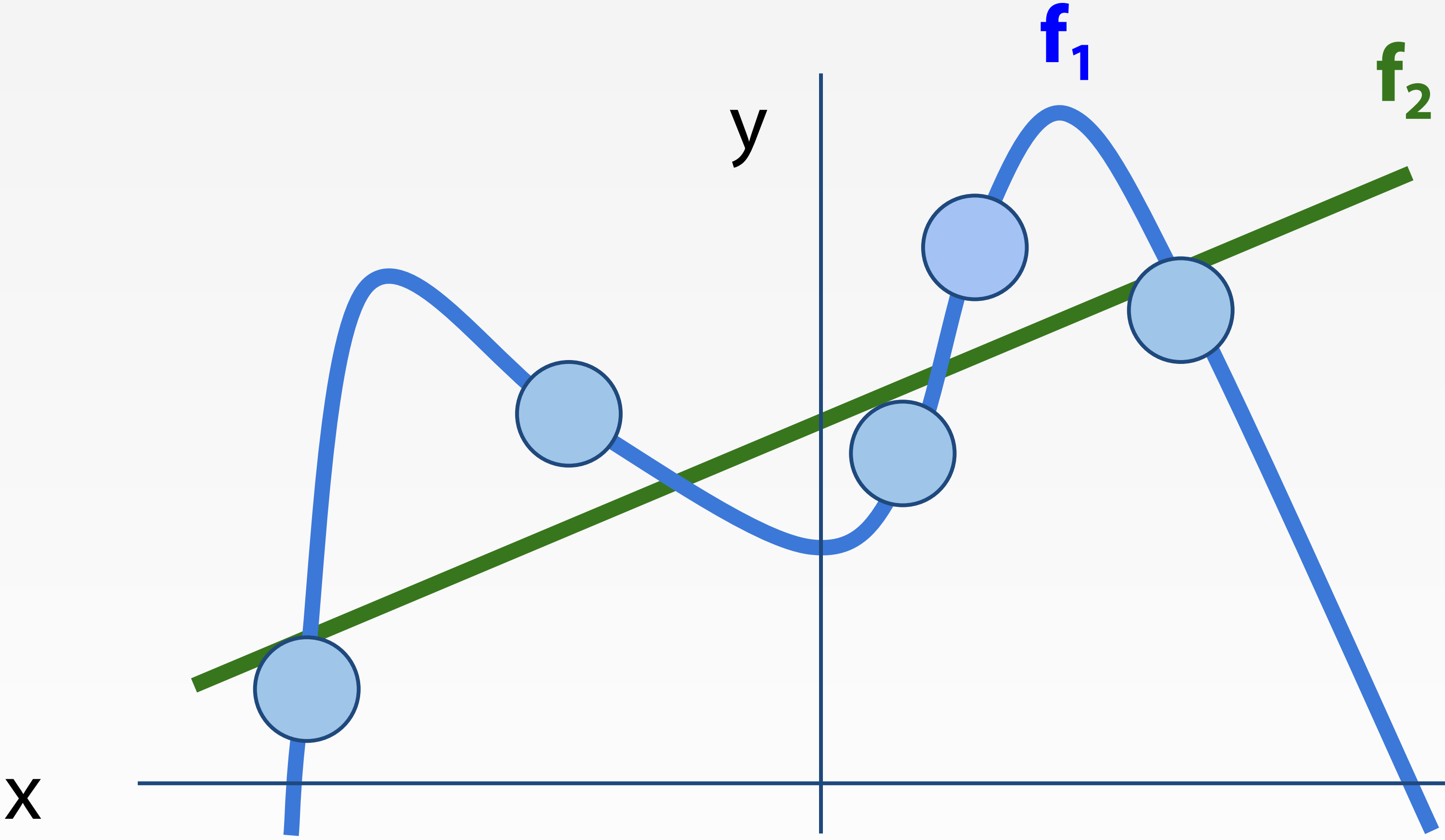
$$R(W) = \sum_k \sum_l W_{k,l}^2$$

L2 regularization likes to "spread out" the weights

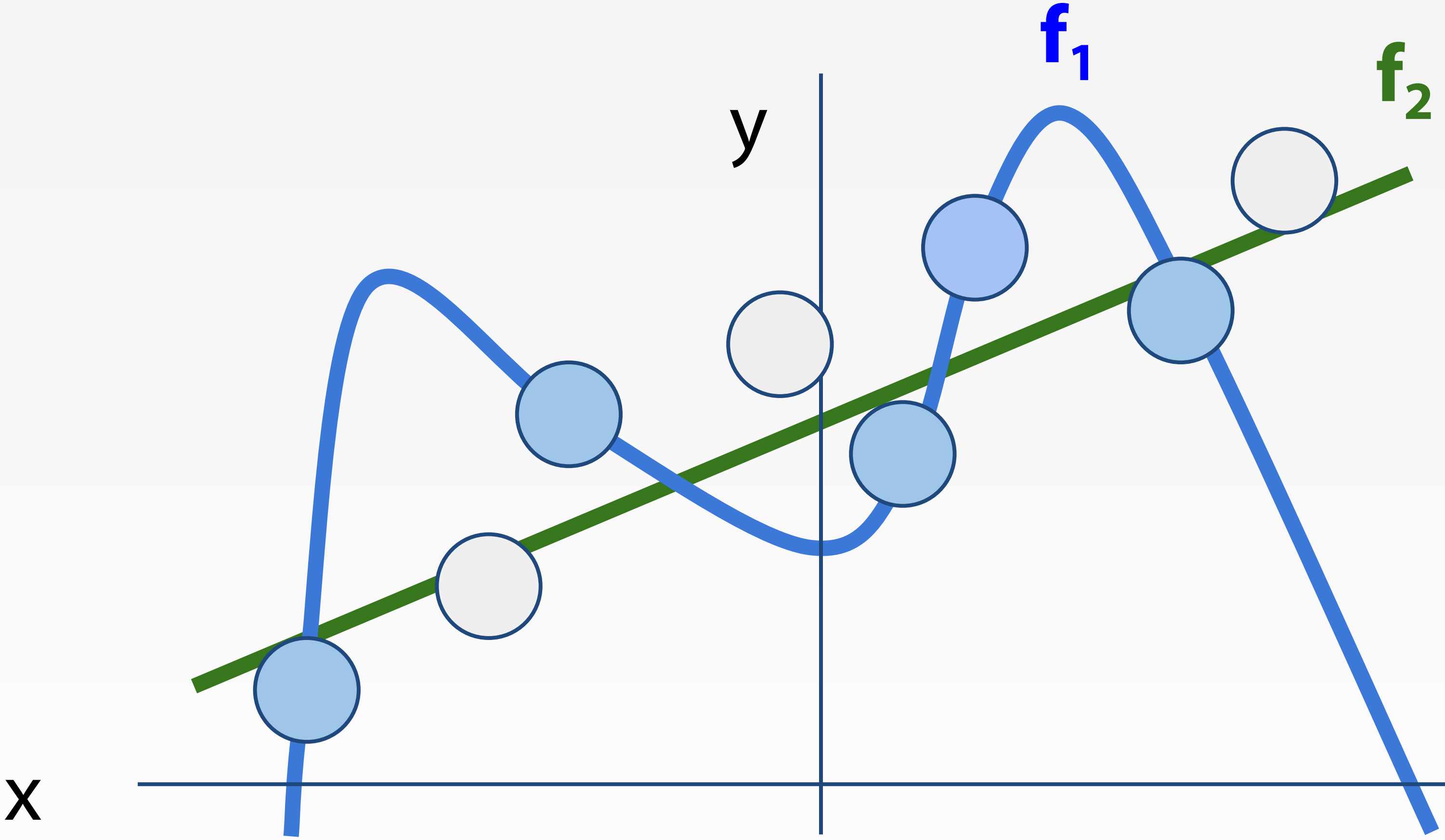
REGULARIZATION: PREFER SIMPLER MODELS



REGULARIZATION: PREFER SIMPLER MODELS



REGULARIZATION: PREFER SIMPLER MODELS



Regularization pushes against fitting the data *too* well so we don't fit noise in the data

SOFTMAX LOSS FUNCTIONS

SOFTMAX LOSS FUNCTION

Want to interpret raw classifier scores as **probabilities**

cat	**3.2**
car	5.1
frog	-1.7

SOFTMAX LOSS FUNCTION

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k|X = x_i) = \boxed{\frac{e^{s_k}}{\sum_j e^{s_j}}}}$$

Softmax Function

cat	**3.2**
car	5.1
frog	-1.7

SOFTMAX LOSS FUNCTION

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

Softmax Function

Probabilities must be >= 0

cat	**3.2**		**24.5**
car	5.1	$\xrightarrow{\text{exp}}$	164.0
frog	-1.7		0.18

unnormalized probabilities

SOFTMAX LOSS FUNCTION

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

cat	**3.2**		**24.5**	**0.13**
car	5.1	$\xrightarrow{\text{exp}}$	164.0	$\xrightarrow{\text{normalize}}$ 0.87
frog	-1.7		0.18	0.00

unnormalized probabilities

probabilities

SOFTMAX LOSS FUNCTION

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

Softmax Function

$$\boxed{L_i = -\log P(Y = y_i|X = x_i)}$$

Probabilities must be >= 0

Probabilities must sum to 1

cat	**3.2**		**24.5**		**0.13**	→ L_i = -log(0.13) = **2.04**
car	5.1	$\xrightarrow{\text{exp}}$	164.0	$\xrightarrow{\text{normalize}}$	0.87	
frog	-1.7		0.18		0.00	

unnormalized probabilities

probabilities

SOFTMAX LOSS FUNCTION

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i|X = x_i)$$

cat	**3.2**	**24.5**	**0.13**
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

exp →

normalize →

unnormalized probabilities

probabilities

→ L_i = -log(0.13)
= **2.04**

Maximum Likelihood Estimation
Choose weights to maximize the likelihood of the observed data

SOFTMAX LOSS FUNCTION

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

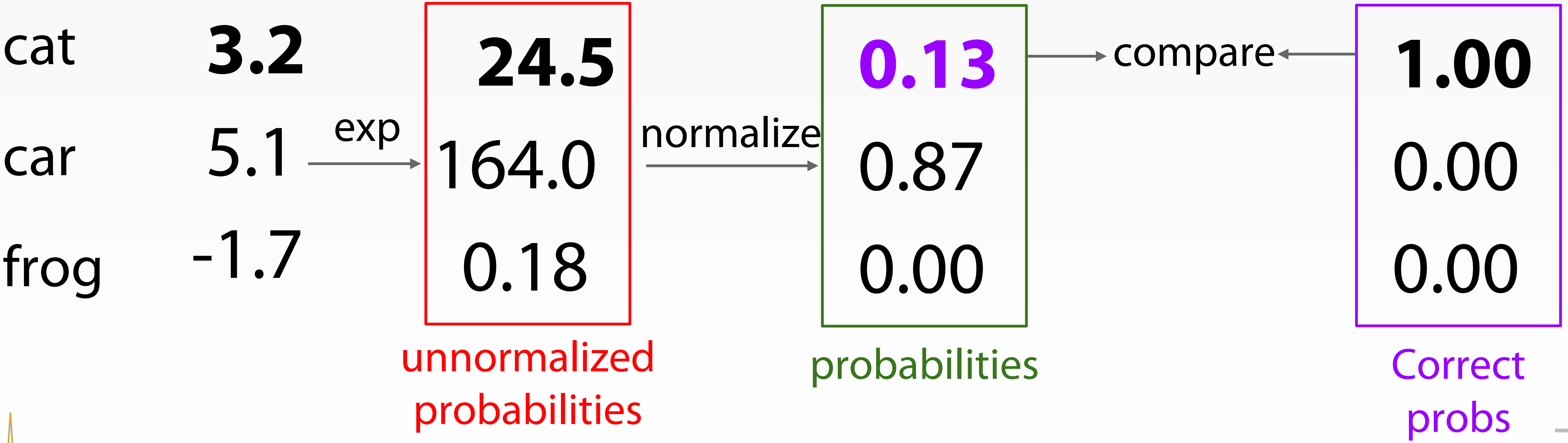
$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

$$L_i = -\log P(Y = y_i|X = x_i)$$

Probabilities must be >= 0

Probabilities must sum to 1



cat	**3.2**	**24.5**	**0.13**	compare	**1.00**
car	5.1	164.0	0.87		0.00
frog	-1.7	0.18	0.00		0.00

unnormalized probabilities

probabilities

Correct probs

SOFTMAX LOSS FUNCTION

Want to interpret raw classifier scores as **probabilities**

$s = f(x_i; W)$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i|X = x_i)$$

cat	**3.2**	**24.5**	**0.13**	**1.00**
car	5.1	164.0	0.87	0.00
frog	-1.7	0.18	0.00	0.00

exp →

normalize →

→ compare ←

Kullback–Leibler divergence

$$D_{KL}(\underline{P}\|\underline{Q}) = \sum_y P(y) \log \frac{P(y)}{Q(y)}$$

unnormalized probabilities

probabilities

Correct probs

SOFTMAX LOSS FUNCTION

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i|X = x_i)$$

			unnormalized probabilities		probabilities		Correct probs
cat	**3.2**		**24.5**		**0.13**	→ compare ←	**1.00**
car	5.1	exp →	164.0	normalize →	0.87	*Cross Entropy*	0.00
frog	-1.7		0.18		0.00		0.00

$$H(\underline{P}, \underline{Q}) = H(p) + D_{KL}(P\|Q)$$

SOFTMAX LOSS FUNCTION

cat	**3.2**
car	5.1
frog	-1.7

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Maximize probability of correct class

$$L_i = -\log P(Y = y_i | X = x_i)$$

Putting it all together:

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SOFTMAX LOSS FUNCTION

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Maximize probability of correct class

$$L_i = -\log P(Y = y_i | X = x_i)$$

Putting it all together:

$$L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

cat	**3.2**
car	5.1
frog	-1.7

Q: What is the min/max possible loss L_i?

SOFTMAX LOSS FUNCTION

cat	**3.2**
car	5.1
frog	-1.7

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

Softmax Function

Maximize probability of correct class

$$\boxed{L_i = -\log P(Y = y_i|X = x_i)}$$

Putting it all together:

$$\boxed{L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})}$$

Q: What is the min/max possible loss L_i?

A: min 0, max infinity

SOFTMAX LOSS FUNCTION

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Maximize probability of correct class

$$L_i = -\log P(Y = y_i|X = x_i)$$

Putting it all together:

$$L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

cat	**3.2**
car	5.1
frog	-1.7

Q2: At initialization all s will be approximately equal; what is the loss?

SOFTMAX LOSS FUNCTION

cat	**3.2**
car	5.1
frog	-1.7

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Maximize probability of correct class

$$L_i = -\log P(Y = y_i | X = x_i)$$

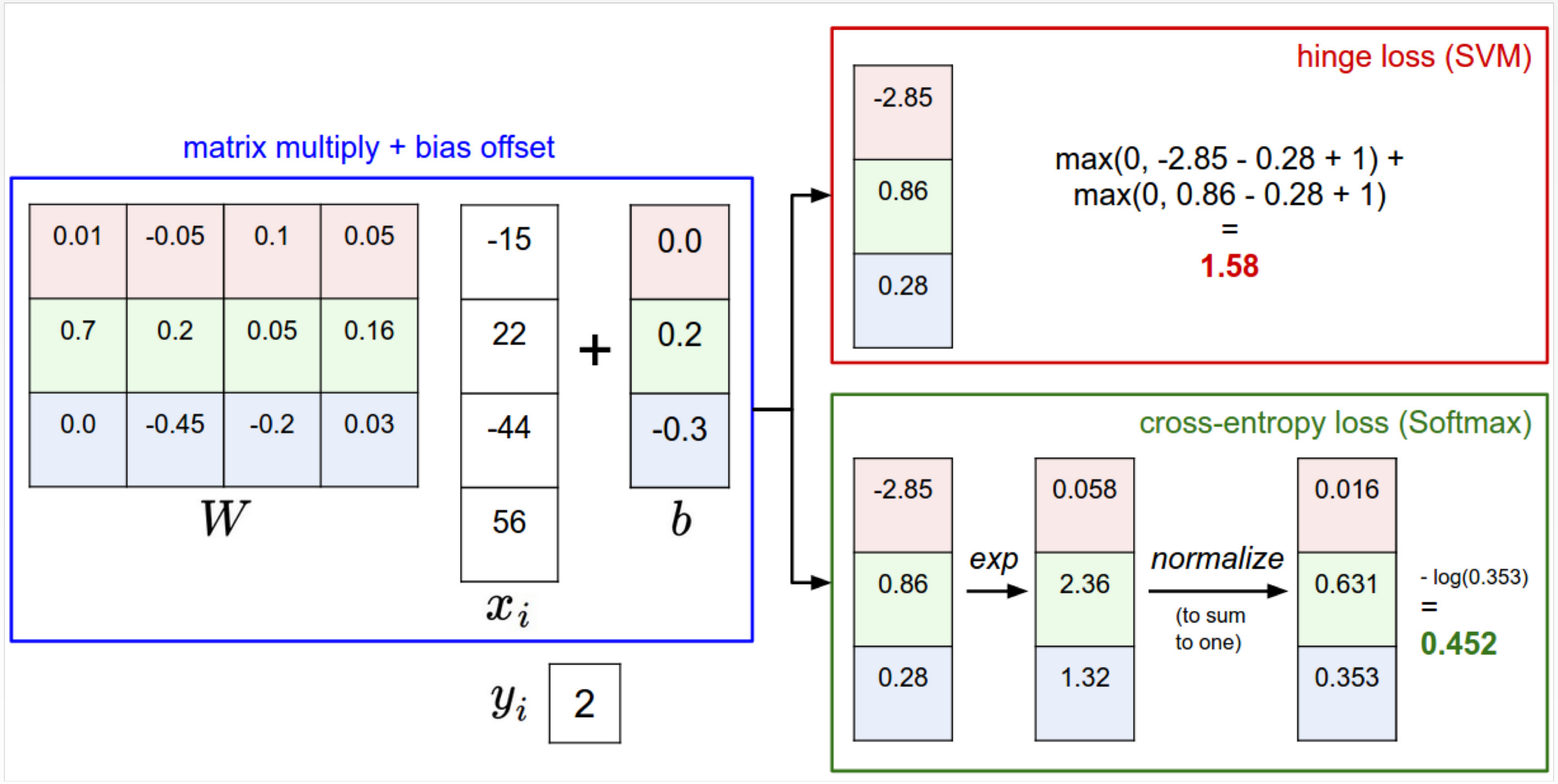
Putting it all together:

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Q2: At initialization all s will be approximately equal; what is the loss?

A: log(C), eg log(10) ≈ 2.3

SOFTMAX VS SVM LOSS FUNCTION



matrix multiply + bias offset

0.01	-0.05	0.1	0.05
0.7	0.2	0.05	0.16
0.0	-0.45	-0.2	0.03

W

-15
22
-44
56

x_i

+

0.0
0.2
-0.3

b

y_i 2

hinge loss (SVM)

-2.85
0.86
0.28

max(0, -2.85 - 0.28 + 1) +
max(0, 0.86 - 0.28 + 1)
=
1.58

cross-entropy loss (Softmax)

-2.85		0.058		0.016
0.86	*exp* →	2.36	*normalize* → (to sum to one)	0.631
0.28		1.32		0.353

- log(0.353)
=
0.452

SOFTMAX VS SVM LOSS FUNCTION

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

SOFTMAX VS SVM LOSS FUNCTION

$$L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

assume scores:

[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and $\boxed{y_i = 0}$

Q: Suppose I take a datapoint and I jiggle a bit (changing its score slightly). What happens to the loss in both cases?

RECAP: LOSS FUNCTIONS

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax

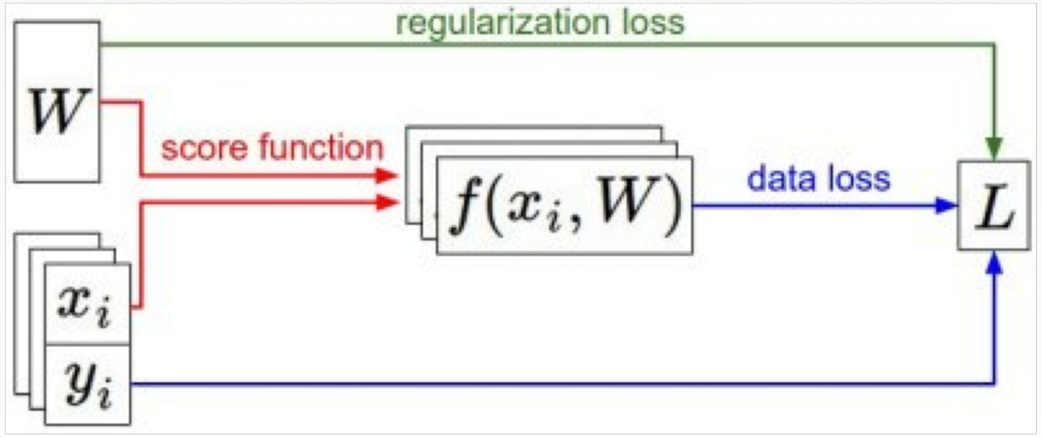
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$

Full loss



RECAP: LOSS FUNCTIONS

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax

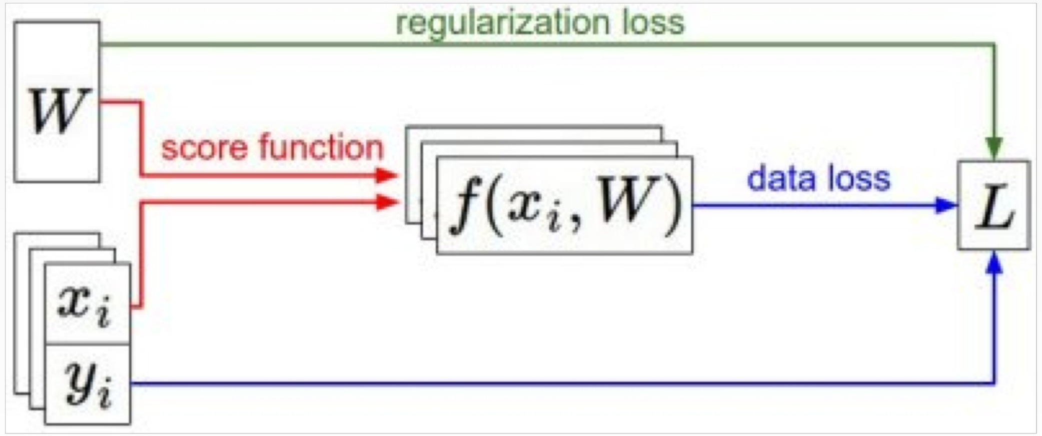
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$ Full loss

How do we find the best W?

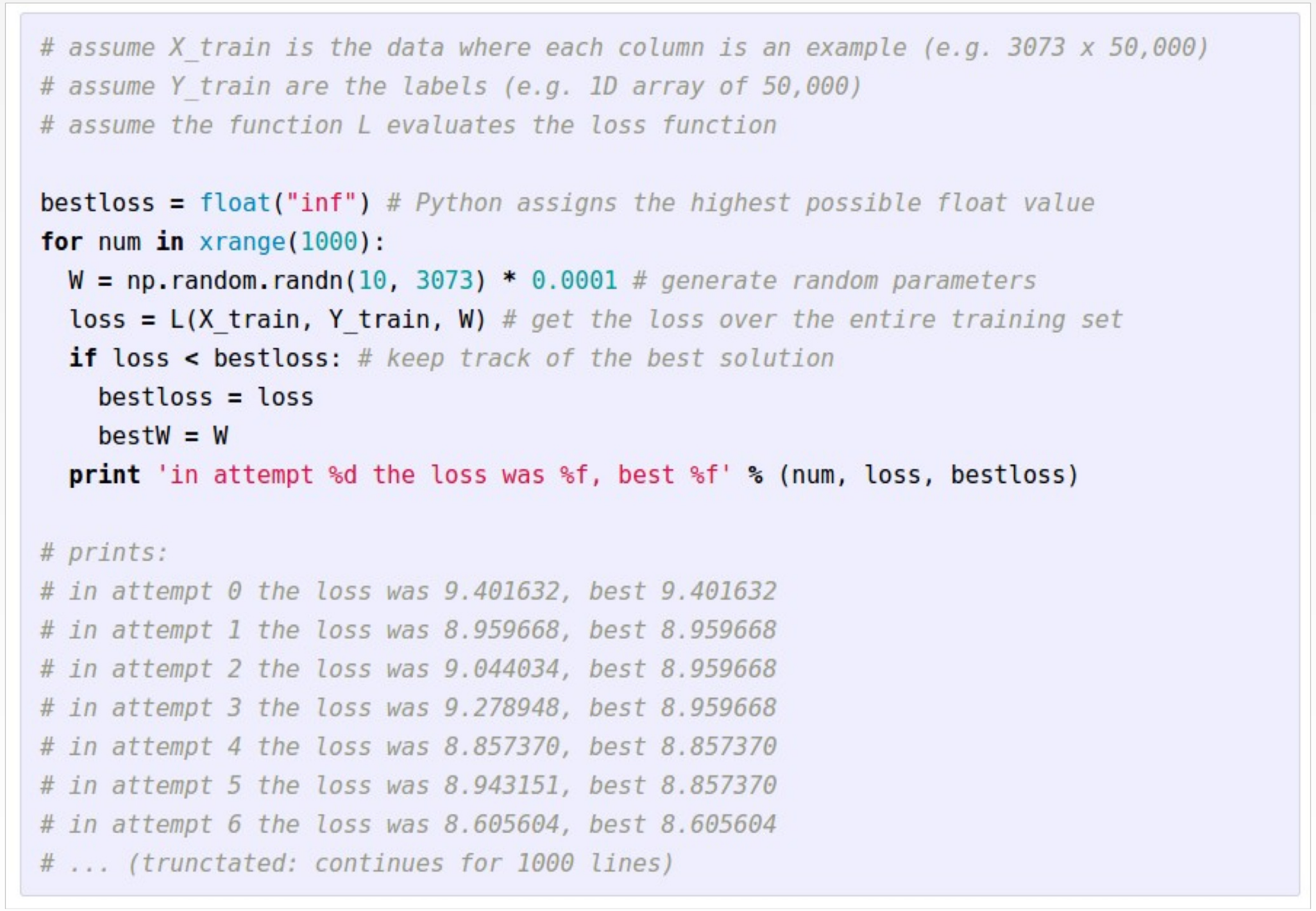


OPTIMIZATION

OPTIMIZATION

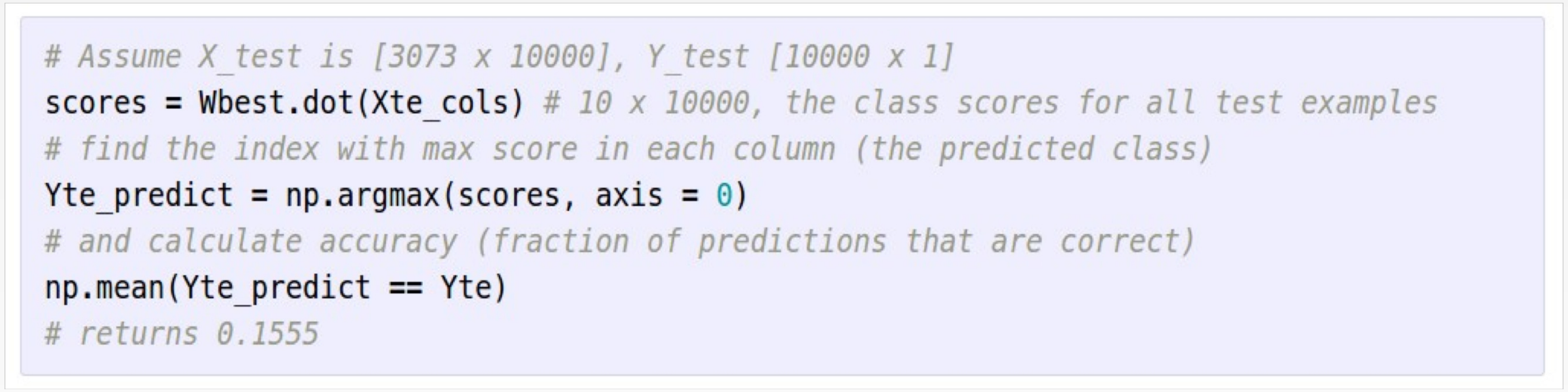
OPTIMIZATION

STRATEGY #1: RANDOM SEARCH



```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in xrange(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (truncated: continues for 1000 lines)
```

STRATEGY #1: RANDOM SEARCH



```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

15.5% accuracy! not bad!
(SOTA is ~95%)

STRATEGY #2: FOLLOW THE SLOPE

STRATEGY #2: FOLLOW THE SLOPE

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension

The slope in any direction is the **dot product** of the direction with the gradient

The direction of steepest descent is the **negative gradient**

current W:	**gradient dW:**
[0.34,	[?,
-1.11,	?,
0.78,	?,
0.12,	?,
0.55,	?,
2.81,	?,
-3.1,	?,
-1.5,	?,
0.33,…]	?,…]
loss 1.25347	

current W:	**W + h** (first dim):	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[?,
-1.11,	-1.11,	?,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

current W:	**W + h** (first dim):	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[**-2.5**,
-1.11,	-1.11,	?,
0.78,	0.78,	?
0.12,	0.12,	
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

(1.25322 - 1.25347)/0.0001
= -2.5

$$\boxed{\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}}$$

current W:	**W + h** (first dim):	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[-2.5,
-1.11,	-1.11,	?,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

current W:	**W + h** (first dim):	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[-2.5,
-1.11,	-1.11,	**0.6**,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

(1.25353 - 1.25347)/0.0001

= 0.6

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

current W:	**W + h** (first dim):	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

current W:	**W + h** (first dim):	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78,	**0**,
0.12,	0.12,	?,
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

(1.25347 - 1.25347)/0.0001
= 0

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

current W:	**W + h** (first dim):	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78,	**0**,
0.12,	0.12,	
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

Numeric Gradient

- Slow! Need to loop over all dimensions
- Approximate

ANALYTIC GRADIENT

- This is silly. The loss is just a function of W:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

- want $\nabla_W L$

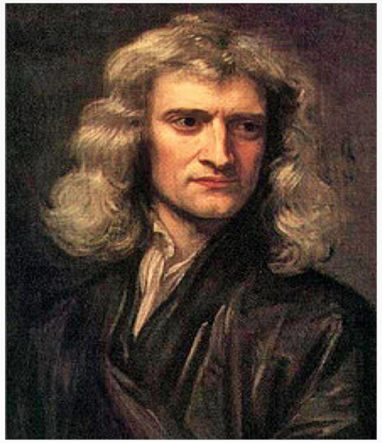
ANALYTIC GRADIENT

- This is silly. The loss is just a function of W:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$



- Want $\nabla_W L$
- Use calculus to compute **analytic gradient**

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

dW = ...
(some function data and W)

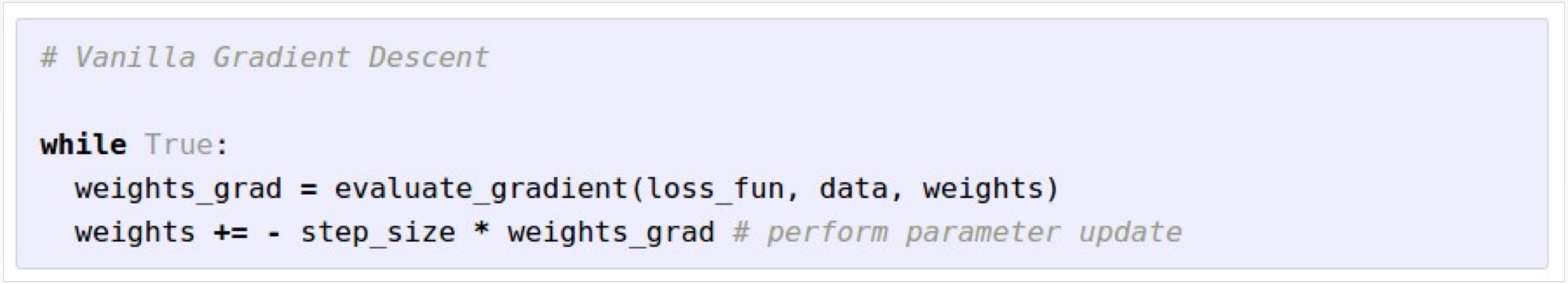
gradient dW:

[-2.5,
0.6,
0,
0.2,
0.7,
-0.5,
1.1,
1.3,
-2.1,…]

SUMMARY

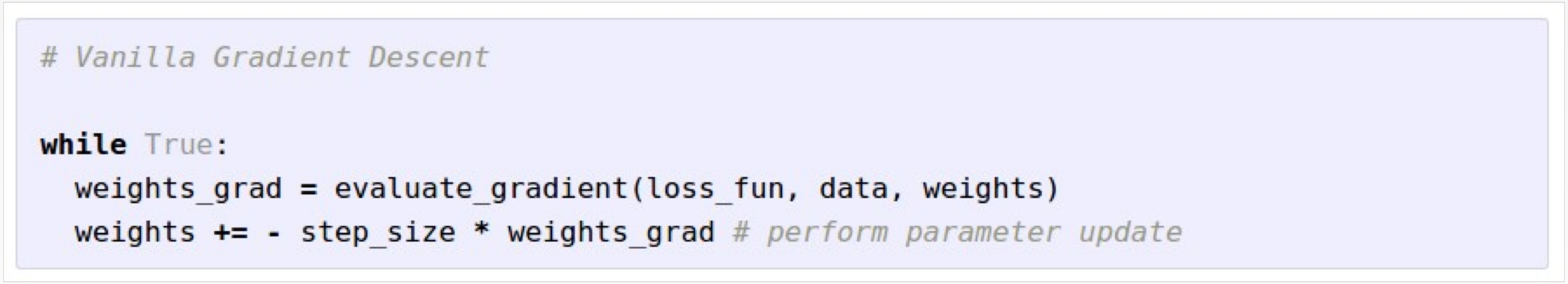
- Types of Gradients
  - Numerical gradient: approximate, slow, easy to write
  - Analytic gradient: exact, fast, error-prone
- In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check.**

GRADIENT DESCENT

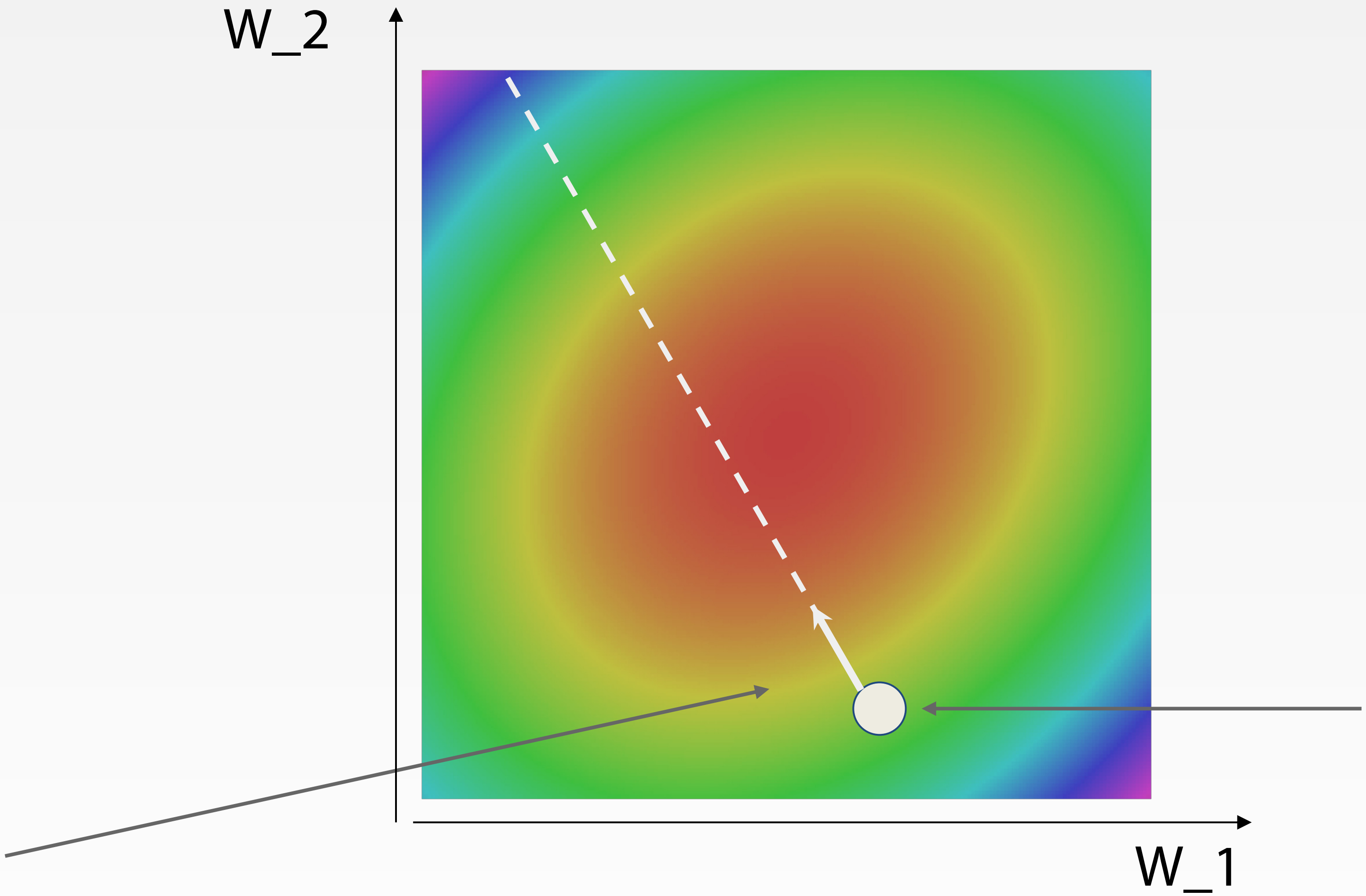


```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

GRADIENT DESCENT



```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```



original W

negative gradient direction

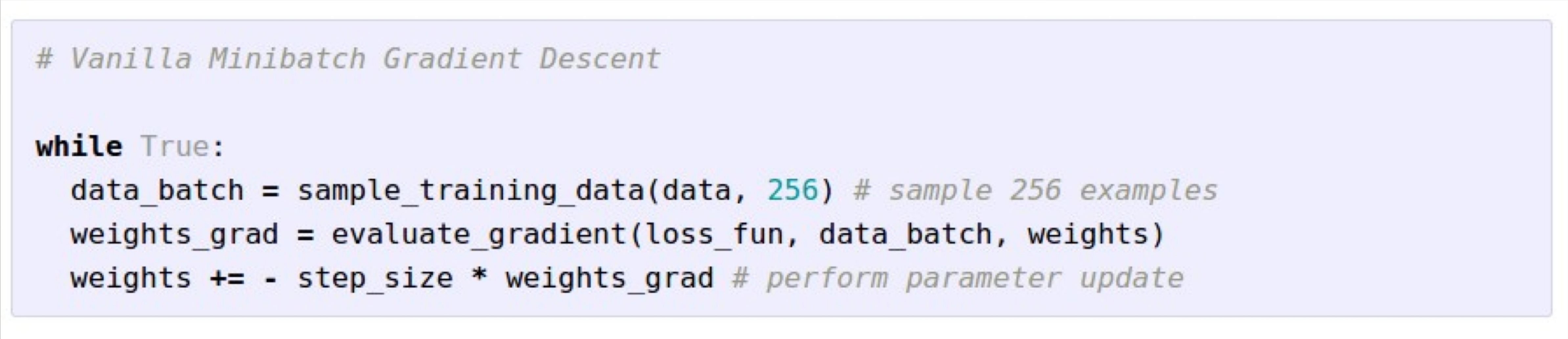
STOCHASTIC GRADIENT DESCENT (SGD)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive when N is large!

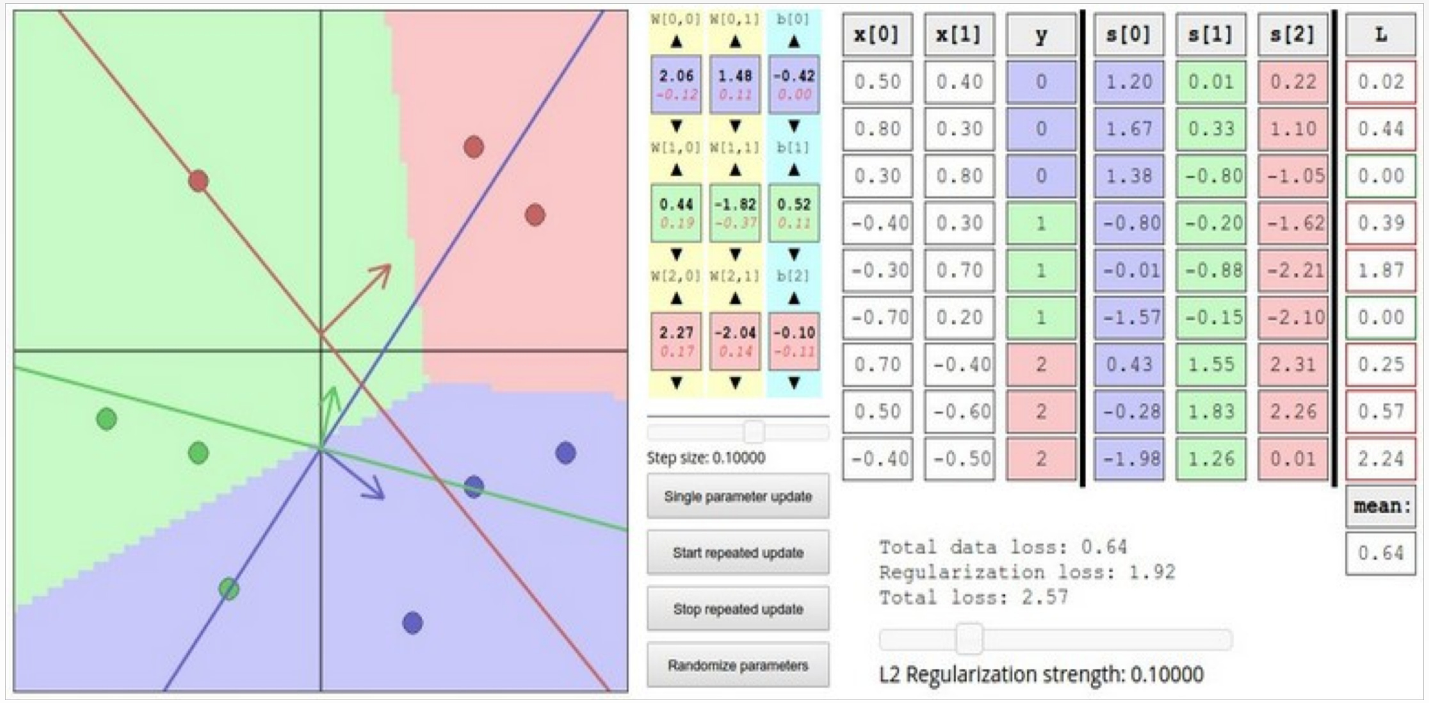
Approximate sum using a **minibatch** of examples
32 / 64 / 128 common



```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

INTERACTIVE WEB DEMO

- http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/



W[0,0] 2.06 -0.12	W[0,1] 1.48 0.11	b[0] -0.42 0.00
W[1,0] 0.44 0.19	W[1,1] -1.82 -0.37	b[1] 0.52 0.11
W[2,0] 2.27 0.17	W[2,1] -2.04 0.14	b[2] -0.10 -0.11

Step size: 0.10000

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	1.20	0.01	0.22	0.02
0.80	0.30	0	1.67	0.33	1.10	0.44
0.30	0.80	0	1.38	-0.80	-1.05	0.00
-0.40	0.30	1	-0.80	-0.20	-1.62	0.39
-0.30	0.70	1	-0.01	-0.88	-2.21	1.87
-0.70	0.20	1	-1.57	-0.15	-2.10	0.00
0.70	-0.40	2	0.43	1.55	2.31	0.25
0.50	-0.60	2	-0.28	1.83	2.26	0.57
-0.40	-0.50	2	-1.98	1.26	0.01	2.24
						mean:
						0.64

```
Total data loss: 0.64
Regularization loss: 1.92
Total loss: 2.57
```

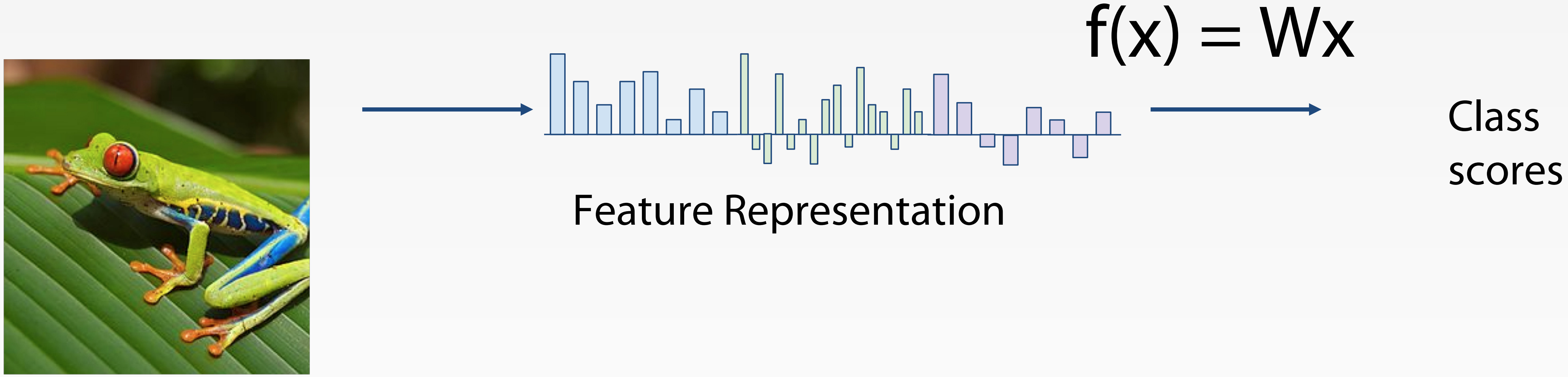
L2 Regularization strength: 0.10000

ASIDE: IMAGE FEATURES

Class scores

$$f(x) = Wx$$

ASIDE: IMAGE FEATURES

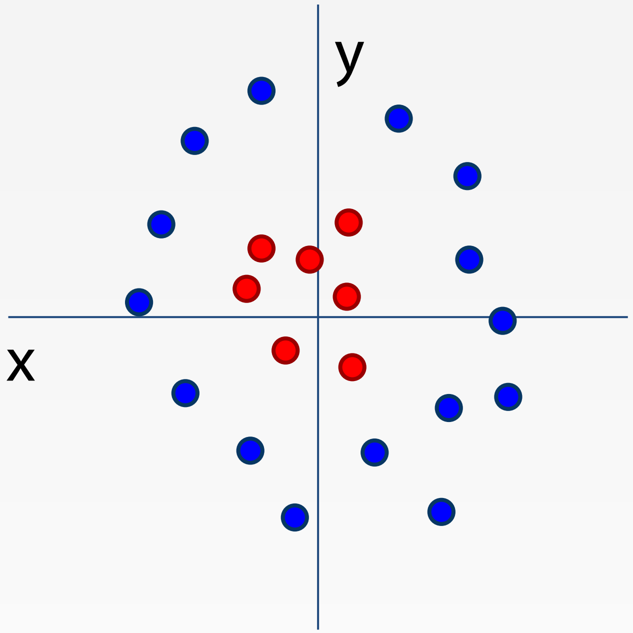


$f(x) = Wx$

Feature Representation

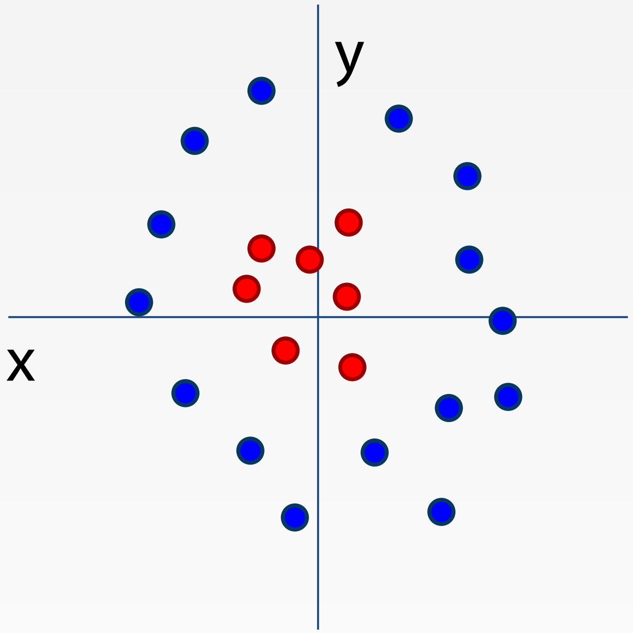
Class scores

IMAGE FEATURES: MOTIVATION

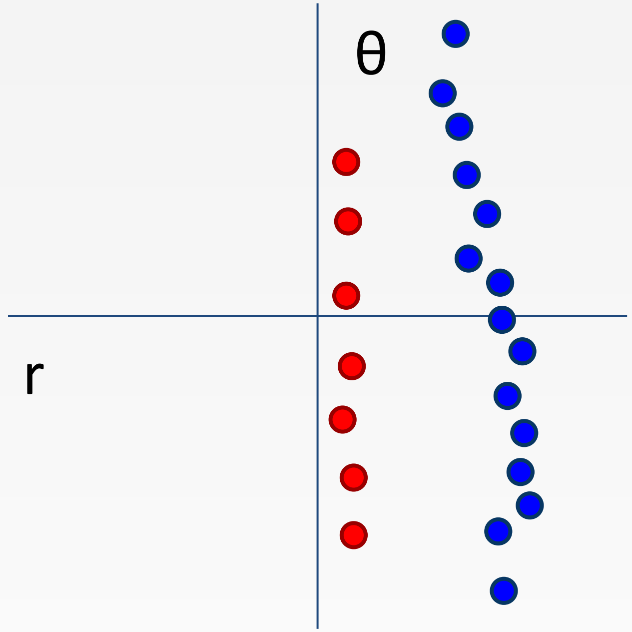


Cannot separate red and blue points with linear classifier

IMAGE FEATURES: MOTIVATION



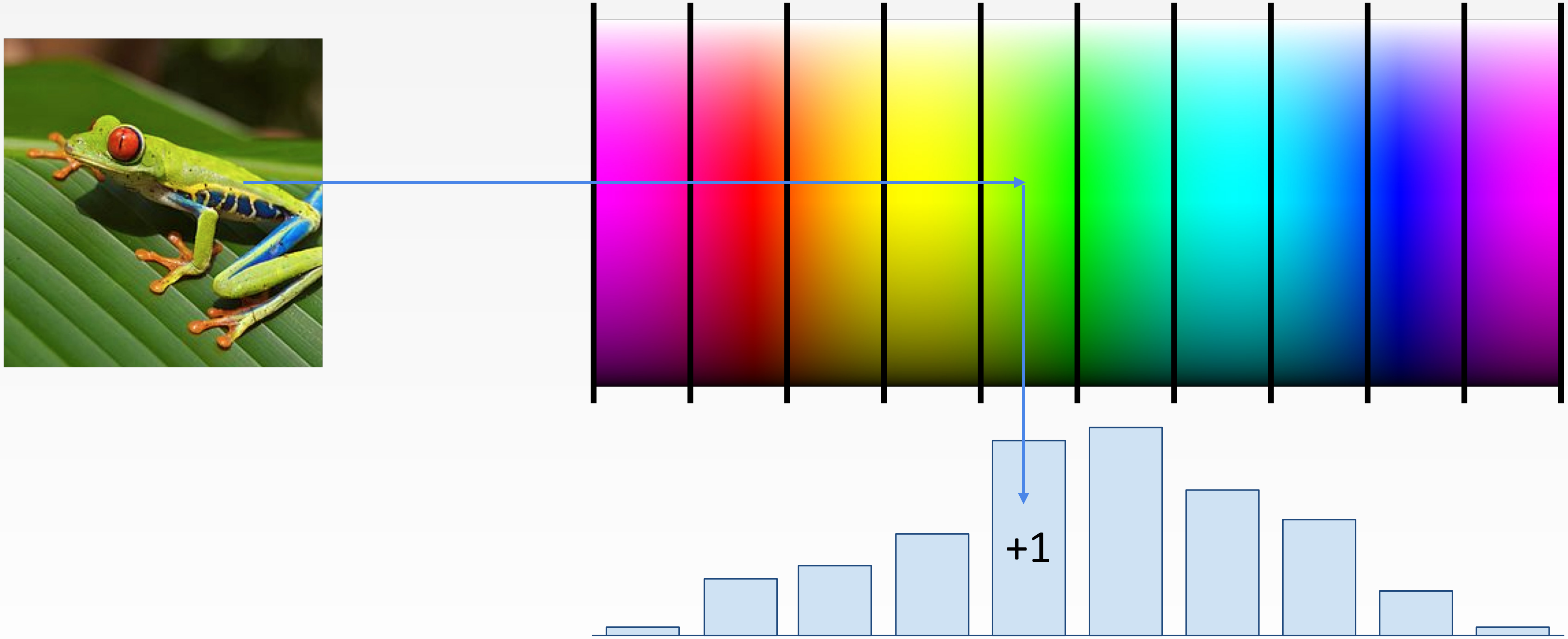
$$f(x, y) = (r(x, y), \theta(x, y)) \longrightarrow$$



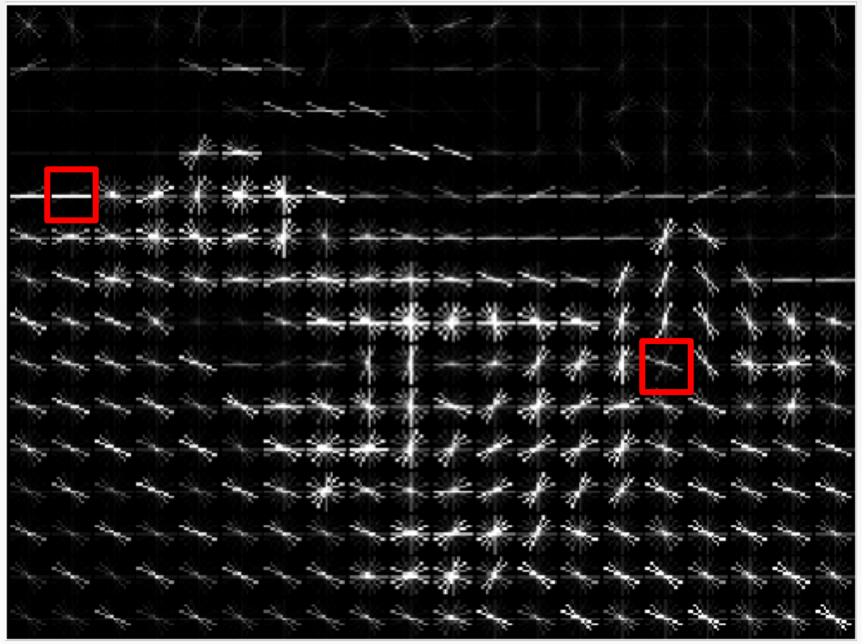
Cannot separate
red and blue points
with linear classifier

After applying feature
transform, points can be
separated by linear classifier

EXAMPLE: COLOR HISTOGRAM



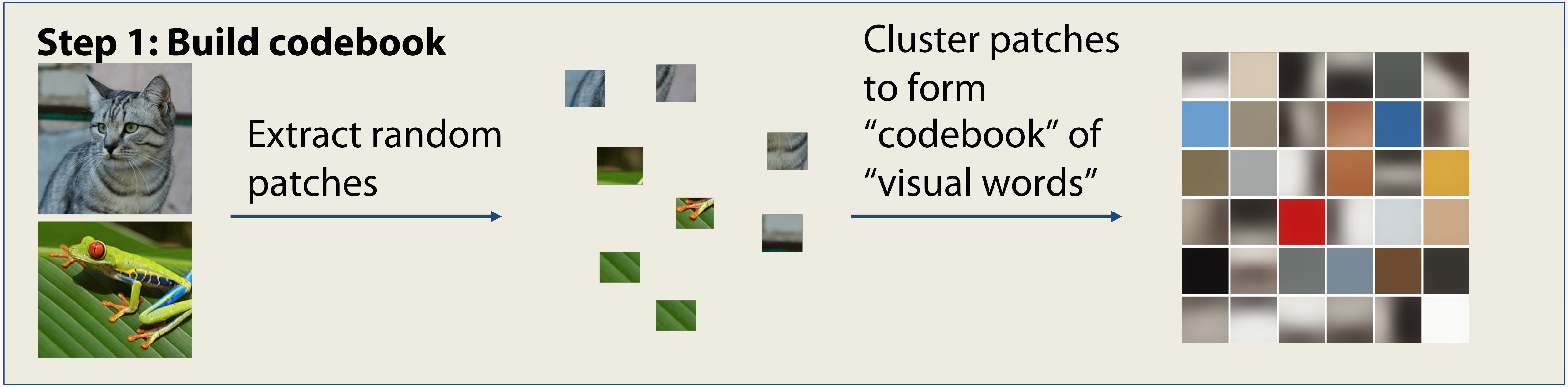
EXAMPLE: HISTOGRAM OF ORIENTED GRADIENTS (HOG)



Divide image into 8x8 pixel regions
Within each region quantize edge direction into 9 bins

Example: 320x240 image gets divided into 40x30 bins;
in each bin there are 9 numbers
so feature vector has 30*40*9 = 10,800 numbers

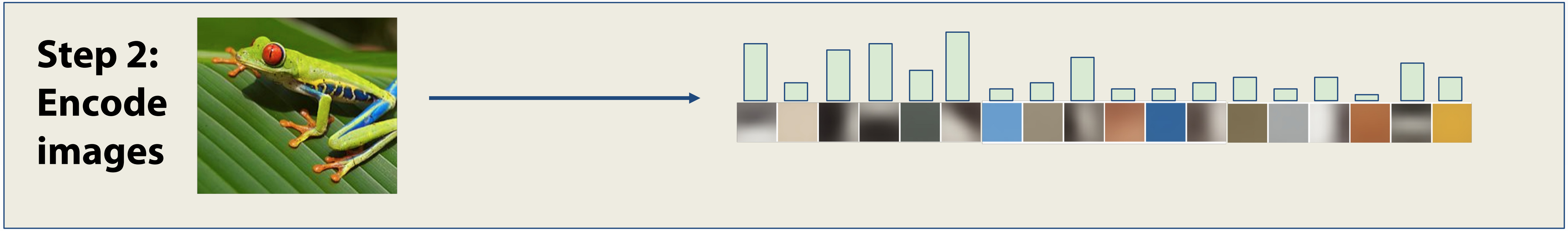
EXAMPLE: BAG OF WORDS



Step 1: Build codebook

Extract random patches

Cluster patches to form "codebook" of "visual words"



Step 2: Encode images

ASIDE: IMAGE FEATURES

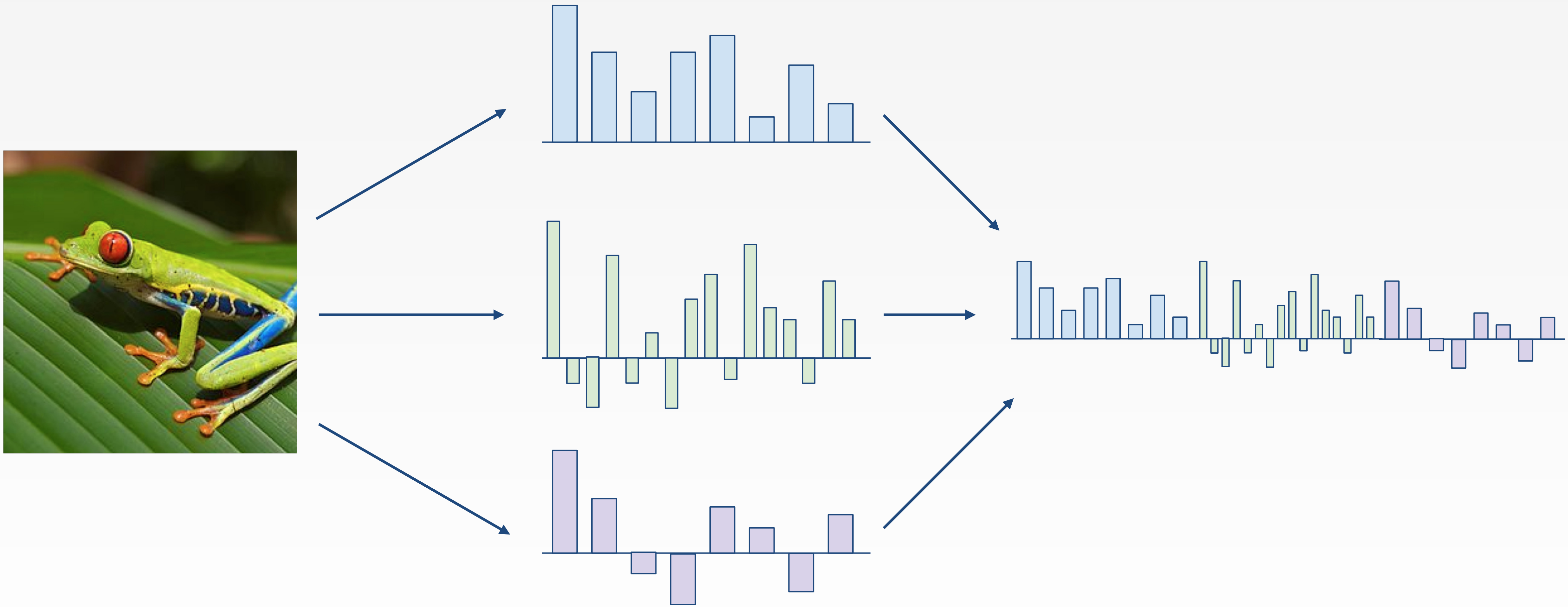
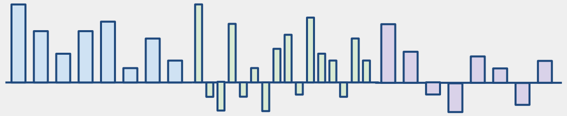


IMAGE FEATURES VS CONVNETS

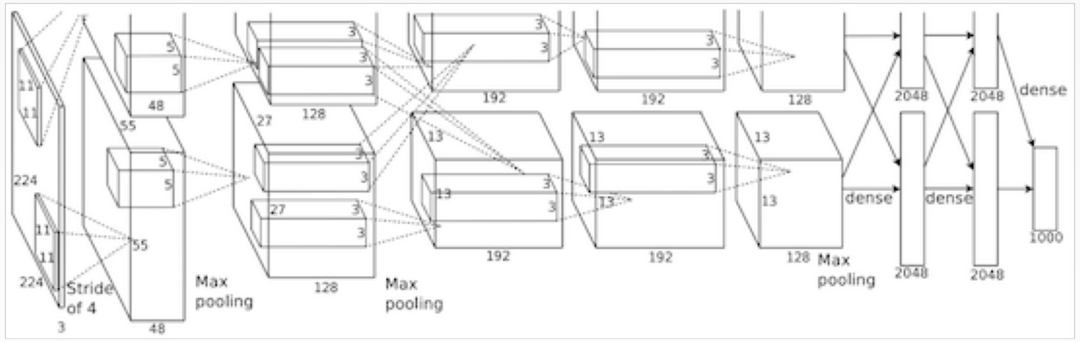
Feature Extraction



f

10 numbers giving scores for classes

training



Krizhevsky, Sutskever, and Hinton, “Imagenet classification with deep convolutional neural networks”, NIPS 2012.
Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

training

10 numbers giving scores for classes

NEXT LECTURE

- Introduction to neural networks
- Backpropagation